1. Spark RPC 实现思路

Spark RPC 的具体实现在源码项目中的 core 子模块中，具体的包路径是： org.apache.spark.rpc

大致思路：

```
1、编写一个服务端 RpcEnv（真正的RPC服务端是：TransportServer），启动一个 Endpoint
2、编写一个客户端 RpcEnv（真正的RPC客户端是：TransportClient），获取一个 EndpointRef
3、然后客户端 EndpointRef 通过三个方法发送消息给服务端的 Endpoint
    1、send 方法
    2、ask 方法
    3、askSync 方法
4、服务端的 HelloEndpoint 进行对应的请求处理，然后返回响应
```

对比理解：

```
1、RpcEnv = ActorSystem
2、Endpoint = Actor
3、EndpointRef = ActorRef
```

Spark 底层的网络通信：

```
代码的设计所用的思路套路是一样的，只不过底层技术支撑发生了改变：akka的代码实现，改成用netty的代码实现
```

spark-1.6.3（akka） spark-2.x（rpc netty）

```
1、RpcEnv
2、Endpoint
3、EndpointRef
4、TransportContext
5、Config
6、TransportServer    真正的Rpc服务端
7、TransportClient    真正的Rpc客户端
......
```

2. Spark RPC 简易代码实现

2.1. Server 端实现

```
package org.apache.spark

import org.apache.spark.rpc.{RpcEndpoint, RpcEnv}
import org.apache.spark.sql.SparkSession

/**************************************************
 * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
 *  注释： Spark RPC 服务端
 */
object RpcServerMain {

    def main(args: Array[String]): Unit = {

        val conf: SparkConf = new SparkConf()

        val sparkSession =
SparkSession.builder().config(conf).master("local[*]").appName("NX
RPC").getOrCreate()
        val sparkContext: SparkContext = sparkSession.sparkContext
        val sparkEnv: SparkEnv = sparkContext.env

        /**
         * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
         *  注释： 构建 RpcEnv
         */
        val rpcEnv = RpcEnv
            .create(HelloRpcSettings.getName(), HelloRpcSettings.getHostname(),
HelloRpcSettings.getHostname(), HelloRpcSettings.getPort(), conf,
                sparkEnv.securityManager, 1, false)

        // TODO_MA 注释：创建 和启动 endpoint
        val helloEndpoint: RpcEndpoint = new HelloEndPoint(rpcEnv)
        rpcEnv.setupEndpoint(HelloRpcSettings.getName(), helloEndpoint)

        rpcEnv.awaitTermination()
    }
}
```

2.2. Client 端实现

```
package org.apache.spark

import org.apache.spark.rpc.{RpcAddress, RpcEndpointRef, RpcEnv}
import org.apache.spark.sql.SparkSession

import scala.concurrent.duration.Duration
import scala.concurrent.{Await, Future}
```

```
/**************************************************
 * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
 *  注释：客户端
 */
object RpcClientMain {

    def main(args: Array[String]): Unit = {
        val conf: SparkConf = new SparkConf()

        val sparkSession =
SparkSession.builder().config(conf).master("local[*]").appName("test
rpc").getOrCreate()
        val sparkContext: SparkContext = sparkSession.sparkContext
        val sparkEnv: SparkEnv = sparkContext.env

        /**
         * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
         *   注释： RpcEnv
         */
        val rpcEnv: RpcEnv = RpcEnv
            .create(HelloRpcSettings.getName(), HelloRpcSettings.getHostname(),
HelloRpcSettings.getPort(), conf, sparkEnv.securityManager, false)

        /**
         * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
         *   注释： endPointRef
         */
        val endPointRef: RpcEndpointRef = rpcEnv
            .setupEndpointRef(RpcAddress(HelloRpcSettings.getHostname(),
HelloRpcSettings.getPort()), HelloRpcSettings.getName())

        import scala.concurrent.ExecutionContext.Implicits.global

        // TODO_MA 注释： one way 异步发送，发送过后既忘
        endPointRef.send(SayHi("test send"))

        // TODO_MA 注释： 不重试的异步发送一次消息， 对应的 endpoint组件通过
receiveAndReply 进行处理
        // TODO_MA 注释： 在超时时间内，返回一个 Future 对象
        val future: Future[String] = endPointRef.ask[String](SayHi("test ask"))
        future.onComplete {
            case scala.util.Success(value) => println(s"Got the Ask result =
$value")
            case scala.util.Failure(e) => println(s"Got the Ask error: $e")
        }
        Await.result(future, Duration.apply("30s"))

        // TODO_MA 注释： 同步发送消息，等待响应，可能等待超时
        val res = endPointRef.askSync[String](SayBye("test askSync"))
        println(res)

        sparkSession.stop()
    }
}
```

2.3. Endpoint 业务载体实现

```
package org.apache.spark

import org.apache.spark.rpc.{RpcCallContext, RpcEndpoint, RpcEnv}

/**************************************************
 * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
 *  注释： RPC 服务组件
 */
class HelloEndPoint(override val rpcEnv: RpcEnv) extends RpcEndpoint {

    // TODO_MA 注释： 在实例被构造出来的时候，自动执行一次
    override def onStart(): Unit = {
        println(rpcEnv.address)
        println("Start HelloEndPoint")
    }

    // TODO_MA 注释： 服务方法
    override def receive: PartialFunction[Any, Unit] = {
        case SayHi(msg) => println(s"Receive Message: $msg")
    }

    override def receiveAndReply(context: RpcCallContext): PartialFunction[Any,
Unit] = {
        case SayHi(msg) => {
            println(s"Receive Message: $msg")
            context.reply(s"I'm Server, $msg")
        }
        case SayBye(msg) => {
            println(s"Receive Message: $msg")
            context.reply(s"I'm Server, $msg")
        }
    }

    override def onStop(): Unit = {
        println("Stop HelloEndPoint")
    }
}

// TODO_MA 注释： 消息类
case class SayHi(msg: String)

// TODO_MA 注释： 消息类
case class SayBye(msg: String)
```

2.4. 配置类 HelloRpcSettings 实现

```
package org.apache.spark

/**************************************************
 * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
 *  注释： 配置类
```

```
 */
object HelloRpcSettings {

    // TODO_MA 注释：  RPC 组件的名称，绑定端口，主机名称等
    val rpcName = "hello-rpc-service"
    val port = 9527
    val hostname = "localhost"

    def getName() = {
        rpcName
    }

    def getPort(): Int = {
        port
    }

    def getHostname(): String = {
        hostname
    }
}
```

3. Spark RPC 代码测试

观看课程的测试！